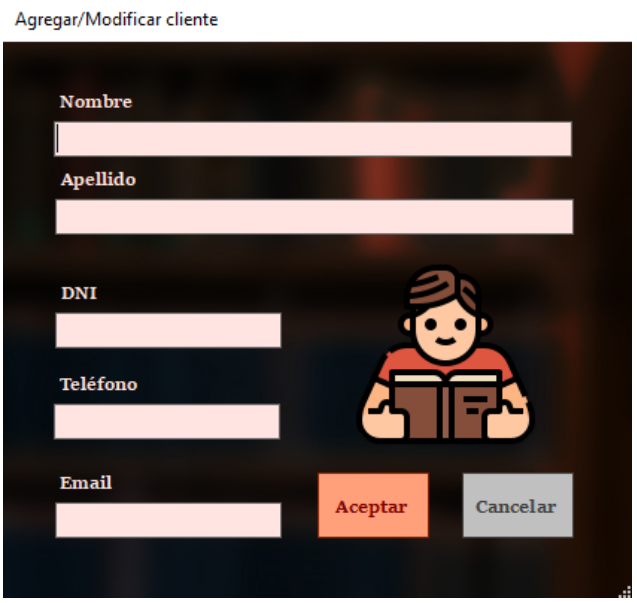
Trabajo práctico N°3

Librería - El rufián melancólico

Este programa está pensado para el empleado de una librería. Puede cargar libros, modificarlos y removerlos, al igual que agregar clientes y modificarlos. Tanto libros como clientes, se encuentran dispuestos en el menú principal. La idea es mantener un registro de los clientes y sus importes totales (podemos generar la compra de un libro y el importe se agregara al cliente especificado) y a la vez, administrar los libros que tiene la librería.



Interfaces

IValidadorDeCampos

Interfaz encargada de la validación de los campos de todos los formularios. Cuenta con dos métodos:

void ValidarCamposVacios()

En cada formulario, este método se encarga de validar que los campos no se encuentren vacíos. De encontrar un campo vacío lanza una excepción del tipo **CampoVacioException** (excepción personalizada que se menciona en el apartado de Excepciones).

bool ValidarCantidadNumeros()

En cada formulario, este método es el encargado de validar que los campos que toman números, los tomen de una cantidad de cifras específicas. Por ejemplo, no se permitirá cargar un cliente cuyo DNI no sea un número de ocho cifras. Retorna true si la cantidad de cifras es la adecuada y false en el caso contrario.

```
5 references
void IValidadorDeCampos.ValidarCamposVacios()
{
    if (this.tb_nombre.Text == String.Empty ||
        this.tb_apellido.Text == String.Empty ||
        this.tb_email.Text == String.Empty ||
        this.tb_dni.Text == String.Empty ||
        this.tb_telefono.Text == String.Empty)
    {
        throw new CampoVacioException("Debes completar todos los campos para continuar");
    }
}

5 references
bool IValidadorDeCampos.ValidarCantidadNumeros()
{
    if (this.tb_dni.Text.Length == 8 && this.tb_telefono.Text.Length == 10)
    {
        return true;
    }
    return false;
}
```

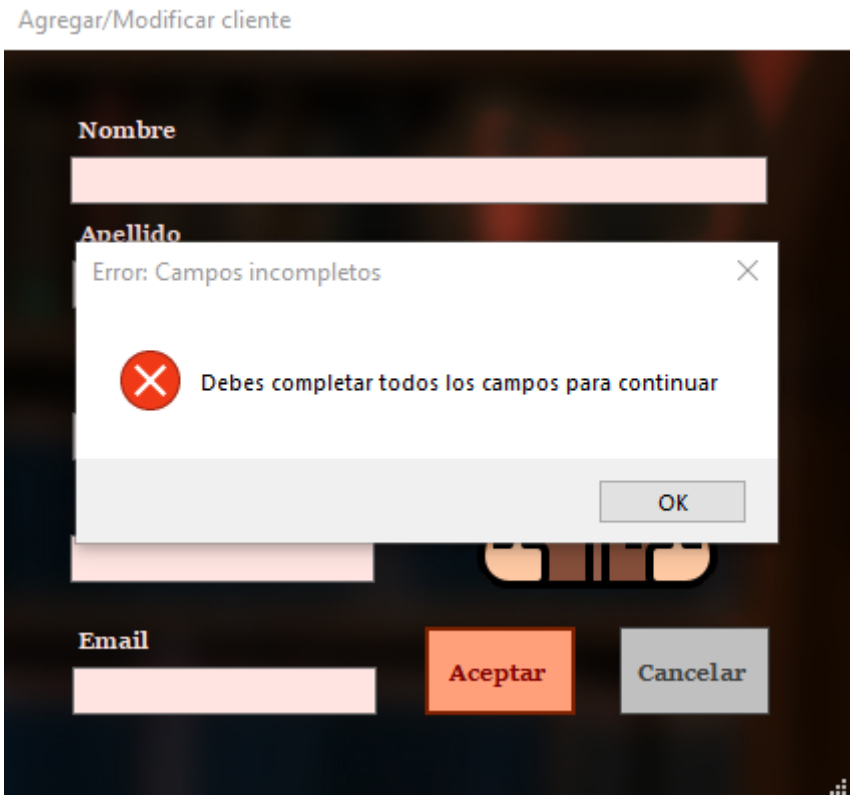
Excepciones

CampoVacioException
Si al ingresar los datos (tanto al agregar como modificar) de un libro o cliente se detecta que hay por lo menos un campo vacío (método **ValidarCamposVacios()** de la interfaz), se lanza la Exception y se muestra un MessageBox con su mensaje.

ClienteExistenteException
Si al ingresar los datos de un cliente (tanto al agregar como modificar), se ingresa el DNI de un cliente ya registrado, se lanza la Exception y se muestra un MessageBox con su mensaje.

LibroExistenteException
Si al ingresar los datos de un libro (tanto al agregar como modificar), se ingresa el DNI de un cliente ya registrado, se lanza la Exception y se muestra un MessageBox con su mensaje.

```
catch (CampoVacioException ex)
{
    MessageBox.Show(ex.Message, "Error: Campos incompletos", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
catch (ClienteExistenteException ex)
{
    MessageBox.Show(ex.Message, "Error: Cliente existente", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
catch (Exception)
{
    MessageBox.Show("Ocurrió un problema al agregar/modificar este cliente", "Error: Fallo en Alta/Modificacion", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```



Serialización y genéricos

SerializadorJson
Clase static que en su implementación trabaja con datos de tipo **List<Libro>** o **List<Cliente>**. Al iniciarse el programa, se cargan los datos de los archivos .js (de existir) encontrados en la carpeta **\Archivos** ubicada en: **\Begnis.Valentin.2A.TP3\Begnis.Valentin.2A.TP3\bin\Debug\net5.0-windows**. Al salir del programa, se guardarán los datos en los archivos.

Archivos

Archivo
Clase static que en su implementación se encarga de escribir en el archivo **Facturas.txt**. Cada vez que se genera una compra, guarda datos del cliente y del libro en una factura que agregara a este archivo ubicado en: **\Begnis.Valentin.2A.TP3\Begnis.Valentin.2A.TP3\bin\Debug\net5.0-windows**.

Tests unitarios

Se realizaron testeos para comprobar las funcionalidades de agregar un cliente a la librería y la igualdad entre dos libros en base a su código.

```
[TestClass]
0 references
public class Tests
{
    [TestMethod]
    0 references
    public void AgregarCliente_CuandoYaExisteUnClienteConElMismoDni_DeberiaRetornarFalse()
    {
        // Arrange
        Libreria libreria = new Libreria();
        Cliente clienteUno = new Cliente("Francisco", "Rodriguez", 40234543, 1134225676, "franrg@gmail.com");
        Cliente clienteDos = new Cliente("John", "Ramson", 40234543, 1145872673, "john12ram@gmail.com");

        // Act
        libreria.ListaClientes.Add(clienteUno);

        bool clienteAgregadoConExito = libreria + clienteDos;

        // Assert
        Assert.IsFalse(clienteAgregadoConExito);
    }

    [TestMethod]
    0 references
    public void LibroExistente_CuandoDosLibrosTienenElMismoCodigo_DeberiaRetornarTrue()
    {
        // Arrange
        Libreria libreria = new Libreria();
        Libro libroUno = new Libro("La metamorfosis", "Franz Kafka", EGenero.Ficcion, 176, 1234, 1500);
        Libro libroDos = new Libro("Los demonios", "Fiódor Dostoyevski", EGenero.Ficcion, 912, 1234, 3800);

        // Act
        libreria.ListaLibros.Add(libroUno);
        bool libroYaExiste = libreria == libroDos;

        // Assert
        Assert.IsTrue(libroYaExiste);
    }
}
```